7

(19) The Japanese Patent Office
(12) Laid-Open Patent Application Publication (A)
(11) Laid-Open Patent Application Publication No. Hei-7-181967
(43) Publication Date: July 21, 1995
(51)

Int. Cl.[5]	Classification Symbol	JPO Ref. No.	FI	
G10H 1/00	101	C Z		
7/02		8938-5H	G10H 7/00	521 F

Request for Examination: not yet requested
Number of Claims: 2 (16 pages in total)

(21) Application No. Hei-5-328841
(22) Application Date: December 24, 1993
(71) Applicant: 000116068
Roland Kabushiki Kaisha
No. 1-4-16, Joshimahama, Kita-ku, Osaka-shi, Osaka, Japan
(72) Inventor: Kenji SATO
No. 1-4-16, Joshimahama, Kita-ku, Osaka-shi, Osaka, Japan
c/o Roland Kabushiki Kaisha
(74) Agent: Patent Attorney, Masaki YAMADA (Another 2 persons)

(54) [Title of the Invention] PLAY DATA READING DEVICE

(57) [Abstract]
[Object]
The present invention relates to a play data reading device for reading play data stored in an external memory such as a floppy disk with improved use efficiency of an internal memory.
[Constitution]
A cache being (or having just been) empty is not loaded fixedly with play data for a corresponding track, but with play data for a track with the smallest number of total clocks (play time) of play data being loaded.

[Claims]

[Claim 1]

A play data reading device for reading play data for plural tracks constituting a song from an external memory mounted, comprising:

plural internal memories for storing the play data read from the external memory in the unit by which the external memory is accessed;

data retrieving means for sequentially retrieving the play data for each track stored in the plural internal memories; and

reading control means for reading, when any of the plural internal memories has become empty, next play data for a track for which play data stored in the plural internal memories will be most immediately finished playing, and subsequent to the play data stored, from the external memory, to store the next play data in the empty internal memory.

[Claim 2]

A play data reading device for reading play data for plural tracks constituting a song from a disk device, comprising:

plural internal memories for storing the play data read from the disk device in the unit by which the disk device is accessed;

data retrieving means for sequentially retrieving the play data for each track stored in the plural internal memories; and

reading control means for determining, when any of the plural internal memories has become empty, a next memory area of the disk device to be accessed based on the respective time when performance based on the play data for each track being stored in the plural internal memories will finish, and based on the current position of a head of the disk device, and for reading play data stored in the memory area from the disk device, to store the play data in the empty internal memory.

[Detailed Description of the Invention]

[0001]

[Field of Industrial Application]

The present invention relates to a play data reading device for reading play data stored in an external memory such as a floppy disk.

[0002]

[Prior Art]

Automatic play devices have been widely used in recent years. An automatic play device is typically configured to receive an external memory such as a floppy disk (hereinafter an external memory is represented by a floppy disk) to be mounted therein, to play based on the play data, representing any song of choice, stored in the floppy disk being mounted.

[0003]

Conventionally, in order for such automatic play devices to play automatically, play data, entirely representing one whole song, stored in a floppy disk being mounted is transferred once to an internal memory such as a RAM, and after the transfer, the play data is sequentially read from the internal memory for performance. This technique, however, requires an internal memory with a storage capacity sufficient for play data for at least one song. Since a recent trend has seen play data for one song increasing more and more, the devices need to have internal memories of correspondingly larger capacities, which is an unfavorable trend in terms of cost.

[0004]

In order to solve this issue, there has been proposed an approach which will be described below. FIG. 27 is a flowchart showing a method of loading play data according to the conventional approach, and FIG. 28 is a diagram illustrating the conventional approach. Here, respective parts of play data necessary to constitute one song, such as play data representing rhythm, that representing melody and that representing bass, are referred to as a "track". Here, also, the floppy disk is accessed by the unit called "sector".

[0005]

As shown in FIG. 28, an automatic play device can play up to five tracks, and each track is provided with two caches, A and B, which are temporary memory where data is stored sector by sector. As shown in step 27_1 of FIG. 27, it is determined whether or not caches A, B for each track are empty, and if either cache is empty, play data is loaded into the empty cache (step 27_2). This process is repeated sequentially for all the tracks (step 27_3).

[0006]

For example, tracks 0 - 4 are each provided with two caches

A, B exclusively for that track. Data is stored in caches A, B, and the data in cache A is read packet by packet to be stored temporarily in a buffer. The data is then outputted from the buffer message by message according to the clock, to a MIDI OUT. When the reading of play data in cache A is complete for example for track 0, then play data in cache B is transferred sequentially packet by packet to the buffer. During that transfer, play data subsequent to that having been stored in cache B is read from the floppy disk to be stored in cache A. Thereafter, caches A and B are used alternately in the same manner. The other tracks, 2 - 5, are handled in the same way. The reason for providing two caches for each track is as follows. One transfer of play data from a floppy disk to a cache requires for example about 0.25 seconds, which corresponds to one eighth note at tempo 120. With only one cache for a track, performance has to be suspended when performance based on the play data stored in the cache has been finished, until subsequent play data is transferred.

[0007]

With this configuration, play data can be loaded from a floppy disk sequentially bit by bit for performance, without the need for an internal memory with a large capacity sufficient for play data corresponding to one whole song stored in a floppy disk to be entirely loaded.

[0008]

[Problem to be Solved by the Invention]

Since one transfer of data requires about 0.25 seconds as described above, the foregoing approach of transferring data bit by bit from a floppy disk to internal caches encounters the need to increase the number of caches per one track in order to play stably and accurately with respect to the clock of, for example, fast-tempo songs where the data rate out of the caches is faster than the data transfer rate into the caches. If the number of caches per one track is increased, however, the total memory capacity required of caches can be enormous in proportion to the maximum number of tracks of the device. On the other hand, some songs use only a smaller number of tracks than the maximum number of tracks of the device, and a number of the caches remain empty during performance of such songs, posing another

issue of poor use efficiency of the memory.

[0009]

In view of the foregoing circumstances, the present invention has an object to provide a play data reading device with improved use efficiency of an internal memory.

[0010]

[Means for Solving the Problem]

In order to achieve the foregoing object, a first play data reading device of the present invention is a play data reading device for reading play data for plural tracks constituting a song from an external memory mounted, including: (1) plural internal memories for storing the play data read from the external memory in the unit by which the external memory is accessed; (2) data retrieving means for sequentially retrieving the play data for each track stored in the plural internal memories; and (3) reading control means for reading, when any of the plural internal memories has become empty, next play data for a track for which play data stored in the plural internal memories will be most immediately finished playing, and subsequent to the play data stored, from the external memory, to store the next play data in the empty internal memory.

[0011]

Also, in order to achieve the foregoing object, a second play data reading device of the present invention includes a disk device as the external memory in the first play data reading device described above, and, as an alternative to the (3) reading control means described above, (4) reading control means for determining, when any of the plural internal memories has become empty, a next memory area of the disk device to be accessed based on the respective time when performance based on the play data for each track being stored in the plural internal memories will finish, and based on the current position of a head of the disk device, and for reading play data stored in the memory area from the disk device, to store the play data in the empty internal memory.

[0012]

[Functions]

Carefully observing play data for plural tracks, amounts of data for respective tracks are not the same; for example, a tempo

track contains little data, while a rhythm track or a track for guitar data have large amounts of data.

[0013]

Hence, according to the first automatic play device of the present invention, an internal memory being (or having just been) empty is not loaded fixedly with data for a corresponding track, but with data for a track with the smallest number of total clocks of data being loaded (i.e. with the shortest play time). This increases the use efficiency of the internal memories, allowing accurate performance with respect to the clock based on dense data even with a small number of internal memories.

[0014]

Also, in loading data stored in a disk device provided as the external memory, if the head of the disk device travels greatly, it takes an accordingly long time to load the data. Therefore, it is preferable to load data in such an order that the head travels least. The second automatic play device of the present invention has been made from this viewpoint. It is basically similar to the first automatic play device described above. However, the next data to be loaded is not fixed to data for a track with the smallest number of total clocks of data being loaded, but is selected from several of data with smallest numbers of clocks based on how little the head travels. This overall can achieve faster loading.

[0015]

[Embodiment]

An embodiment of the present invention will be described hereinafter. FIG. 1 is a block diagram showing the configuration of an embodiment of a play data reading device of the present invention. An automatic play device 10 includes a CPU 11 for executing a program, a ROM 12 for storing the program to be executed by the CPU 11, and a RAM 13 used as a work area for the program. In this embodiment, plural caches, as called in the present invention, are secured in the RAM 13.

[0016]

The automatic play device 10 also includes a switch 14 for allowing commands such as start (PLAY) and stop (STOP) of performance, a display 15 for displaying the status of the

automatic play device 10 such as the number of the song or measure being played, a floppy disk control device 17 for controlling a floppy disk device 16 to access a floppy disk mounted in the floppy disk device 16, and a MIDI communication device 18 for outputting play data from a MIDI OUT terminal. The components 11 - 15, 17 and 18 are connected each other via a bus 19. The play data outputted from the MIDI communication device 18 is inputted into, for example, an automatic play device connected externally, which plays automatically based on the inputted play data. Here, the automatic play device is described as being connected to the play data reading device. It is apparent, however, that the automatic play device may include therein the play data reading device of this embodiment.

[0017]

FIG .2 is a flowchart and FIG. 3 is a diagram illustrating the data flow, both showing the basic concept of a data transfer process in this embodiment. As shown in FIG. 3, five caches are provided in this embodiment. In step 2_1 of the routine of FIG. 2, it is determined whether or not any cache i (i = 0 - 4) is empty. If there is any empty cache, a search is made for a track where the total number of clocks of loaded data is the smallest, and the data in that track is loaded into the empty cache in step 2_2. This process is repeated for all the caches (step 2_3). That is, there is no fixed relationship between the caches and the tracks; each time an empty cache emerges, play data for the track where the total number of clocks of loaded data is the smallest, or in other words play data for the track where the loaded data will most immediately run out, is transferred to the empty cache.

[0018]

The data transfer will be described in detail below. FIG. 4 shows the data structure of play data for a song about to be played, stored in a floppy disk mounted in the floppy disk device 16 (see FIG. 1), in this embodiment. Three tracks are used to play this song. Play data for track 1 is stored in sector 0 and a part of sector 1 of the floppy disk. The numbers of clocks (play time) of play data for track 0 in these sectors are respectively 150, 10. Play data for track 1 is stored in a part of sector 1, sectors 2, 3, and a part of sector 4, with the numbers

of clocks for track 1 in these sectors being respectively 10, 50, 50, 50. Likewise, play data for track 2 is stored in a part of sector 4, and sectors 5, 6, with the numbers of clocks for track 2 in these sectors being respectively 20, 100, 40.

[0019]

FIGs. 5 - 14 schematically show the procedure for transferring (loading) play data with the data structure shown in FIG. 4 stored in a floppy disk to the five caches shown in FIG. 3, and changes in values of various flags and registers, used in a data transfer routine executed by the CPU 11 shown in FIG. 1, associated with the data transfer.

[0020]

The roles of the various flags and registers shown in FIGs. 5 - 14 are as follows:

·LoadPtr[i]: a register for storing the number of a cache (0 - 4) where play data for track i (i = 0 - 2) has been loaded last,

·LoadTime[i]: a register for storing the total number of clocks of play data for track i having so far been loaded into caches,

·SectCnt[i]: a register for storing the number of remaining sectors for track i not yet to be loaded into a cache,

·PlayNo[i]: a register for storing the number of a cache where next play data for track i to be outputted is stored,

·Cache[j]: a register for storing the number of a sector of a floppy disk where play data having been loaded into cache j was stored,

·Free[j]: a flag indicating whether cache j is empty ('1') or not ('0'), and

·Next[j]: a register for storing the number of a cache where play data for a track subsequent to play data in the same track stored in cache j is stored. If there is no subsequent play data, the register stores '-1'.

[0021]

Descriptions will be made in order below with reference to FIGs. 5 - 14. FIGs. 5 and 6 illustrate data transfer in a preparatory stage before performance starts. At first, as shown in FIG. 5, all the caches are empty, and play data from sectors 0, 1, 4, or the respective head data for tracks 0, 1, 2, are sequentially transferred respectively to caches 0, 1,

2. This causes LoadPtr[0], LoadPtr[1], LoadPtr[2] to store 0, 1, 2, or the respective number of a cache where data has been loaded, and causes LoadTime[0], LoadTime[1], LoadTime[2] to respectively store '150', '10', '20', or the total number of clocks of play data for respective track 0, 1, 2 having been transferred from a sector. It also causes SectCont[0], SectCont[1], SectCont[2] to respectively store '1', '3', '2', or the number of sectors for respective track 0, 1, 2 where play data yet to be transferred to a cache is stored.

[0022]

It further causes PlayNo[0], PlayNo[1], PlayNo[2] to respectively store '0', '1', '2', or the number of a cache from which play data for performance for the respective track is retrieved. It still further causes Cache[0], Cache[1], Cache[2] to respectively store '0', '1', '4', or the number of a sector of the floppy disk where play data having been transferred to the respective cache was stored. Since no data has yet been transferred to caches 3, 4, Cache[3], Cache[4] are empty. Since some data has been transferred to caches 0, 1, 2, Free[0] - Free[2] now all store '0'. Since caches 3, 4 are empty, Free[3], Free[4] store '1'. Since the play data now stored in caches 1, 2, 3 are at the moment the last play data having been transferred for tracks 0, 1, 2, respectively, Next[0] - Next[2] store '-1', indicating that the data being stored is the last. Next[3], Next[4] also store '-1', as their default.

[0023]

Then, LoadTime[0] - LoadTime[2] are compared to cause play data for a track with the smallest number of clocks (at this time track 1) to be transferred to empty cache 3. Since cache 4 still remains empty, LoadTime[0] - LoadTime[2] are compared again, when play data has been transferred to cache 3, to cause play data for a track with the smallest number of clocks (now track 2) to be transferred to empty cache 4. Consequently, play data has been transferred to all caches 0 - 4.

[0024]

FIG. 6 shows the state at this time. Now, LoadPtr[1], LoadPtr[2] respectively store '3', '4', or the number of a cache where data for the corresponding track has been last

transferred; LoadTime[1], LoadTime[2] respectively store '10 + 50', '20 + 12'0, or the total number of clocks of play data for respective track 1, 2 having been transferred; and SectCont[1], SectCont[2] respectively store '2', '1', or the number of sectors for respective track 1, 2 where untransferred play data is stored.

[0025]

Also, Cache[3], Cache[4] respectively store '2', '5', or the number of a sector where play data stored in that cache was stored; and Free[3], Free[4] store '0', indicating that any data is stored. Further, Next[1], Next[2] respectively store '3', '4', or the number of a cache where subsequent play data is stored.

[0026]

After play data is loaded into all caches 0 - 4 in this manner, performance starts. FIG. 7 shows the state where a period corresponding to 10 clocks has elapsed after performance started. With 10 clocks having elapsed, LoadTime[0], LoadTime[1], LoadTime[2] have now reduced from the state of FIG. 6 by 10 clocks, respectively, to '140', '0 + 50', '10 + 100'. Performance with play data stored in cache 1 ends, which causes changes as follows: Free[1] = 1, Next[1] = -1, PlayNo[1] = 3.

[0027]

Cache 1 is now empty. Thus, for tracks where SectCnt[0] - SectCnt[2] is not '0' (at this time the register is not '0' for any of tracks 0 - 2), LoadTime[0] - LoadTime[2] are compared to load subsequent play data for a track with the smallest number of clocks (track 1) into cache 1.

[0028]

FIG. 8 shows the state where play data for track 1 has been loaded into cache 1. Play data from sector 3 has been transferred to cache 1, which causes changes as follows: Cache[1] = 3, Free[1] = 0, Next[3] = 1, LoadPtr[1] = 1, LoadTime[1] = 50 + 50, SectCont[1] = 1.

[0029]

FIG. 9 shows the state where 10 clocks have elapsed since the state of FIG. 8. With 10 clocks having elapsed, LoadTime[0], LoadTime[1], LoadTime[2] have now reduced from the state of FIG. 8 by 10 clocks, respectively, to '130', '40 + 50', '0 + 100'.

Performance with play data for track 2 stored in cache 2 ends, which causes changes as follows: Free[2] = 1, Next[2] = -1. Also, PlayNo[2] is changed to 4, which causes performance with play data for track 2 stored in cache 4 to start.

[0030]

Cache 2 is now empty. Thus, for tracks where SectCnt[0] - SectCnt[2] is not '0', LoadTime[0] - LoadTime[2] are compared to load subsequent play data for a track with the smallest number of clocks, track 1, into cache 2, as shown in FIG. 10. Subsequent play data for track 1, or play data from sector 4, has been transferred to cache 2, which causes changes as follows: Free[2] = 0, Next[1] = 2, LoadPtr[1] = 2, LoadTime[1] = 40 + 50 + 50, SectCnt[1] = 0.

[0031]

With SectCnt[1] = 0, the play data for track 1 is now all loaded into caches. FIG. 11 shows the state where additional 40 clocks have elapsed since the state shown in FIG. 10. With 40 clocks having elapsed, LoadTime[0], LoadTime[1], LoadTime[2] have now reduced from the state of FIG. 10 by 40 clocks, respectively, to '90', '0 + 50 + 50', '60'.

[0032]

Play data for track 1 having been stored in cache 3 is now emptied, which causes changes as follows: Free[3] = 1, Next[3] = -1, PlayNo[1] = 1. This in turn causes performance with play data for track 1 stored in cache 1 to start. Cache 3 is now empty. Thus, for tracks where SectCnt[0] - SectCnt[2] is not '0', LoadTime[0] - LoadTime[2] are compared to load subsequent play data for a track with the smallest number of clocks, track 2, into cache 3.

[0033]

FIG. 12 shows the state where subsequent play data for track 2, or play data from sector 6, has been loaded into cache 3. Changes are made as follows: Cache[3] = 6, Free[3] = 0, Next[4] = 3, LoadPtr[2] = 3, LoadTime[2] = 60 + 40, SectCnt[2] = 0.

[0034]

FIG. 13 shows the state where additional 50 clocks have elapsed since the state of FIG. 12. With 50 clocks having elapsed, LoadTime[0], LoadTime[1], LoadTime[2] have now reduced from the state shown in FIG. 12 by 50 clocks, respectively, to 40, 0 +

150, 10 + 40.

[0035]

Play data for track 1 having been stored in cache 1 is now emptied, which causes changes as follows: Free[1] = 1, Next[1] = -1, PlayNo[1] = 2. Cache 1 is now empty. Thus, for tracks where SectCnt[0] - SectCnt[2] is not '0', LoadTime[0] - LoadTime[2] are compared. However, since SectCnt[1] = 0, SectCnt[2] = 0, or in other words the loading of play data for tracks 1, 2 is already complete, play data in remaining track 0 is loaded into cache 1.

[0036]

FIG. 14 shows the state where subsequent play data for track 0, or play data from sector 1, has been transferred to cache 1. Changes are made as follows: Cache[1] = 1, Free[1] = 0, Next[0] = 1, LoadPtr[0] = 1, LoadTime[0] = 40 + 10, SectCnt[0] = 0.

[0037]

With 10 clocks having elapsed thereafter, changes are made as follows: LoadTime[0] = 30 + 10, LoadTime[1] = 40, LoadTime[2] = 0 + 40. This causes performance with play data for track 2 stored in cache 4 to end, which causes changes as follows: Free[4] = 1, Next[4] = -1, PlayNo[2] = 3. With additional 30 clocks having elapsed thereafter, changes are made as follows: LoadTime[0] = 0 + 10, LoadTime[1] = 10, LoadTime[2] = 10. This causes play data for track 0 stored in cache 0 to be emptied, which causes changes as follows: Free[0] = 1, Next[0] = -1, PlayNo = 1.

[0038]

With still additional 10 clocks having elapsed, performance with play data for this one whole song all ends. A description will be made of the data format of play data stored in a floppy disk, which will be followed by descriptions of various routines associated with the play data transfer described with reference to FIGs. 5 - 14.

[0039]

FIG. 15 shows the format of play data stored in a floppy disk. The format includes a one-byte header representing the total number of tracks. The header is followed by data for the respective tracks, including leading four bytes representing

the length (or the number of bytes) of that track, and the rest subsequent thereto representing actual play data.

[0040]

FIG. 16 shows the format of play data. Each play data includes a two-byte time information portion and a three-byte data information portion. FIG. 17 shows the detail of a data information portion included in play data. MIDI information of three bytes is represented as it is; MIDI information of two bytes or one byte is supplemented with an FFh. A data information portion can also represent time information only, tempo, beat, ending information, etc.

[0041]

Routines of FIGs. 18 - 26 will be described next. Since a specific example of data transfer has already been described in detail with reference to FIGs. 5 - 14, the routines will be described only briefly below, however. The roles of important registers and flags used in the routines of FIGs. 18 - 26 will collectively be described below, including those already described.

·Cache[MAX_CACHE][512]: a register for storing the number of a cache where play data is loaded from a disk.

·PlayNo[MAX_TRACK]: a register for storing the number of a cache where play data next to be outputted is stored.

·PlayAddr[MAX_TRACK]: a register for storing the address in a cache of play data next to be outputted.

·TimeCount[MAX_TRACK]: a waiting time counter.

·LoadPtr[MAX_TRACK]: a register for storing the number of a cache where data has been loaded last.

·LoadTime[MAX_TRACK]: a register for storing the number of clocks of play data for each track having already been loaded into caches.

·SectCnt[MAX_TRACK]: a register for storing the number of remaining sectors where play data is stored, for the purpose of determining whether or not the loading for each track is complete.

· EndFlag[MAX_TRACK]: a flag indicating whether or not performance of that track is already complete.

·Next[MAX_CACHE]: a register for storing the number of a next cache.

·Free[MAX_CACHE]: a register indicating whether or not that cache is empty (with '1' representing emptiness).

·SetData[trk][3]: a register for storing play data to be outputted for each track.

·work[5]: a work register used to separate five-byte play data into a time information portion and a data information portion.

·work0: a work register for temporarily storing the total number of clocks, for the purpose of searching for a track where the total number of clocks of loaded data is the smallest.

·Song: a register where the number of a song is stored.

·EndCount: a register for determining the end of a song.

·Buff: a work register for checking the total number of tracks.

·TrkNo: a register for storing the number of tracks of that song.

·trk: a track loop counter 1.

·trk0: a track loop counter 2.

·trk1: a register where the number of a track with the smallest number of loaded clocks (LoadTime).

·i: a loop counter.

·MeasCount: a register for counting the number of a measure.

·Measure: a register where the number of a measure is stored.

FIG. 18 is a flowchart of a main routine which starts being executed when the power turns on.

[0042]

In step 18_1, it is determined whether or not a floppy disk is in the floppy disk device. Although it is not shown in the drawings, when a floppy disk is ejected, the clock stops and all the music is erased, and the process jumps to step 18_1. In step 18_2, the register Song for storing the number of a song is set to '0'. In this embodiment, assuming for simplicity that performance of songs always starts with the song of number 0, initialization is performed in step 18_3. That is, PlayNo, PlayAddr, TimeCount, LoadPtr, SectCnt, LoadTime store '0', and EndFlag stores '1' for all the tracks; Next stores '0' and Free stores '1' for all the caches; and MeasCount, Measure store '0'.

[0043]

In step 18_4, a song indicated by the register Song for storing the number of a song is prepared for performance. The detail will be described later. In step 18_5, a preparation is made on the measure counter. The detail will be described later.

Preparation for performance is now complete. In step 18_6, the process waits for the PLAY switch to be pressed.

[0044]

When the PLAY switch is pressed, the process proceeds to step 18_7, where a command is given to generate a clock interrupt. In step 18_8, it is determined whether or not the STOP switch has been pressed. If it has not been pressed, the process proceeds to step 18_9, where the register EndCount indicating the end of a song is checked. If it is still in the middle of a song (EndCount > 0), the process proceeds to step 18_10.

[0045]

In step 18_10, a command is given to load data from the disk into a cache (Cache), and in step 18_11, a command is given to display the number of a measure. The details of steps 18_10, 18_11 will be described later. On the other hand, if the STOP switch has been pressed, or if the song has come to an end, the process proceeds to step 18_12, where a command is given to perform the processes associated with the pressing of the STOP switch. The detail will be described later.

[0046]

In step 18_13, Song is increased by 1, and in step 18_4, it is determined whether or not Song is equal to or less than the number of songs recorded in the floppy disk. If there still remains a song yet to be played in the floppy disk, the process returns straight to step 18_3. If the last song has already been played, Song is set to '0' to return to the first song, and the process returns to step 18_3.

[0047]

As described above, performance is made cyclically based on the play data stored in the floppy disk. FIG. 19 is a flowchart of a subroutine for preparation for play of a song indicated by Song (see FIG. 5), executed in step 18_4 of FIG. 18. At first, in step 19_1, one sector at the head of a file of that song (see FIG. 15) is loaded into the work register Buff.

[0048]

In step 19_2, the register TrkNo for storing the number of tracks of that song and the register EndCount for determining the end of a song are set to the number of tracks indicated in the header. In step 19_3, the track loop counter trk is set

to '0'.

[0049]

In step 19_4, for the purpose of detecting the end of loading, it is calculated over how many sectors the data indicated by track trk are recorded, to set the calculated value in SectCnt[trk]. In step 19_5, play data from a sector where play data at the head of a track indicated by trk is stored is loaded into Cache[trk].

[0050]

In step 19_6, SectCnt[trk] is decreased by 1, since loading has occurred in step 19_5. In step 19_7, it is determined how many clocks of play data has been loaded in step 19_5, and the obtained number of the clocks is set in the register LoadTime[trk] for storing the number of clocks of play data having so far been loaded into caches.

[0051]

In step 19_8, the register LoadPrt[trk] for storing the number of a cache where data has been last loaded is set to trk. In step 19_9, the register PlayNo[trk] for storing the number of Cache where next play data to be outputted is stored is set to trk, and the register PlayAddr[trk] for storing the address in a cache of play data next to be outputted is set to the address of the head of the play data.

[0052]

In step 19_10, a command is given to set next play data. The detail will be described later. In step 19_11, trk is increased by '1', and in step 19_12, it is determined whether or not trk is equal to or less than TrkNo. If trk is equal to or less than TrkNo, the process returns to step 19_4, and if trk is more than TrkNo, the process proceeds to step 19_13. In step 19_13, a command is given to load play data, in the same manner as in step 18_10 of the main routine shown in FIG. 18. The routine of step 19_13 will be described in detail later.

[0053]

FIG. 20 is a flowchart of a subroutine for setting next play data (see FIG. 6), executed in step 19_10 of the routine of FIG. 19. In step 20_1, PlayNo[trk] is set in a work register no, and PlayAddr[trk] is set in a work register addr.

[0054]

In step 20_2, the work loop counter i is set to '0'. In step 20_3, it is determined whether or not i is equal to or less than 5. This is related to the fact that the play data, constituted of five bytes as shown in FIG. 16 and read byte by byte, requires five times of reading. If i < 5, the process proceeds to step 20_4, where it is determined whether or not addr is equal to or less than 512 (or the number of bytes in one sector). If addr = 512, the process proceeds to step 20_5, where PlayNo[trk] is set to the register Next[no] storing the number of a cache where play data next to be played is loaded, PlayAddr[trk] is set to '0', Next[no] is set to '-1' indicating that there is no next play data, and Free[no] is set to '1' indicating that that cache is empty.

[0055]

In step 20_6, the work register work[i] is set to the cache Cache[no][addr], in step 20_7, i is increased by '1', and the process returns to step 20_3. In step 20_3, if it is determined that i = 5, the process proceeds to step 20_8, where the waiting time counter TimeCount[trk] is increased by additional time.

[0056]

In step 20_9, the register SetData for storing play data to be outputted is set with new data. FIG. 21 is a flowchart of a data loading subroutine for searching for an empty cache and loading play data for a track with the least time of loaded data into the empty cache found, executed in step 18_10 of the main routine shown in FIG. 18 and in step 19_13 of the subroutine shown in FIG. 19.

[0057]

At first, in step 21_1, the work counter i indicating the number of a cache is set to '0'. In step 21_2, the register Free[i] indicating whether or not that cache is free is checked. If it is free, the process proceeds to step 21_2, and if it is not free, the process proceeds to step 21_14.

[0058]

Subsequent steps 21_3 - 21_8 perform the process of searching for a track with the least time of loaded data. In step 21_3, the work register trk0 indicating the number of a track is set to '0', the work register trk1 indicating the number of a track with the smallest number of loaded clocks is set to 0, and the

work register work0 for searching for the smallest number of clocks is set to the maximum time MAX_TIME which can be handled by the present device.

[0059]

In step 21_4, it is determined whether or not the loading completion register SectCnt[trk0] is positive. If it is positive, the process proceeds to step 21_5, and if it is not, the process proceeds to step 21_7. In step 21_5, it is determined whether or not work0 exceeds LoadTime[trk0]. If work0 > LoadTime[trk0], the process proceeds to step 21_6, and if not, the process proceeds to step 21_7.

[0060]

In step 21_6, the work register work0 is set to LoadTime[trk0], and trk1 is set to trk0. In step 21_7, trk0 is increased by '1'. In step 21_8, it is determined whether or not trk0 is equal to or less than TrkNo. If trk < TrkNo, the process returns to step 21_4, and if trk = TrkNo, the process proceeds to step 21_9.

[0061]

In step 21_9, it is determined whether or not work0 has been changed from MAX_TIME having been stored in step 21_3. If it has not been changed, the process ends, and if it has been changed, the process proceeds to step 21_10. Step 21_10 and the subsequent steps perform the process associated with loading data. In step 21_10, next play data for a track indicated by trk1 is loaded into Cache[i].

[0062]

In step 21_11, SectCnt[trk1] is decreased by '1'. In step 21_12, it is determined how many clocks of play data has been loaded in step 21_10, and the value is added to LoadTime[trk1]. In step 21_13, for the purpose of updating link information related to loading, LondPtr[trk1] for storing the number of a cache where data has been last loaded is set in Next[i], LondPtr[trk1] is set to 1, and Free[i] is set to '0'.

[0063]

In step 21_14, i is increased by '1'. In step 21_15, it is determined whether or not i is equal to or less than the number of caches MAX_CACHE. If there is any cache where no processing has been performed, the process proceeds to step 21-2, and if the processing is complete with all the caches, the routine

exits.

[0064]

FIG. 22 is a flowchart of a clock interrupt routine for giving commands to output play data, set next data, update a counter for loaded data, and stop performance automatically after detecting the end of a song. This clock interrupt routine is started at predetermined intervals by an interrupt clock which is generated on receiving the command given to generate a clock interrupt in step 18_7 of the main routine of FIG. 18.

[0065]

When this routine is started, at first, the track counter trk is set to '0' in step 22_1. In step 22_2, the register EndFlag[trk] indicating whether or not performance of that track is already complete is checked. If it is complete, the process proceeds to step 22_8, and if it is not, the process proceeds to step 22_3.

[0066]

In step 22_3, TimeCount[trk] is decreased by '1'. In step 22_4, it is determined whether TimeCount[trk] is positive or not more than 0. If it is positive, the process proceeds to step 22_7, and if it is not more than 0, the process proceeds to step 22_5.

[0067]

In step 22_5, a command is given to output data. The detail of step 22_5 will be described later. In step 22_6, a command is given to set next play data. For detail, see FIG. 20 which has already been described.

[0068]

In step 22_7, LoadTime[trk] is decreased by '1'. In step 22_8, it is determined whether EndCount is positive or 0. If it is positive, the process proceeds to step 22_10, and if it is 0, the process proceeds to step 22_9. In step 22_9, a command is given to perform a play stop process for automatic stop. The detail will be described later.

[0069]

In step 22_10, trk is increased by '1'. In step 22_11, it is determined whether or not trk is equal to or less than TrkNo. If trk < TrkNo, the process returns to step 22_2, and if trk = TrkNo, the process proceeds to step 22_12. In step 22_12,

the register MeasCount for measuring the number of a measure is increased by '1'.

[0070]

In step 22_13, it is determined whether or not MeasCount is equal to or more than 16. If MeasCount > 16, the process proceeds to step 22_14, and if not, the process ends. In step 22_14, MeasCount is set to '1', and the register Measure for storing the number of a measure is increased by '1'.

[0071]

FIG. 23 is a flowchart of a subroutine for outputting play data to a MIDI OUT terminal, updating the tempo and beat, and detecting the end of a song, depending on the type of that play data, executed in step 22_5 of the clock interrupt routine shown in FIG. 22. In step 23_1, it is determined whether or not DataSet[trk] is MIDI information. If it is MIDI information, the process proceeds to step 23_2, where DataSet[trk] is outputted from the MIDI OUT terminal.

[0072]

In step 23_3, it is determined whether or not DataSet[trk] is tempo information. If it is tempo information, the process proceeds to step 23_4, where the tempo is changed. In step 23_5, it is determined whether or not DataSet[trk] is beat information. If it is beat information, the process proceeds to step 23_6, where the beat is changed.

[0073]

In step 23_7, it is determined whether or not DataSet[trk] is ending information. If it is ending information, the process proceeds to step 23_8, where EndCount is decreased by '1', and EndFlag[trk] is set to '0'. FIG. 24 is a flowchart of a subroutine for stopping performance, executed in step 18_12 of the main routine shown in FIG. 18.

[0074]

In step 24_1, a clock interrupt is stopped, and in step 24_2, all the musical sound being produced is muted. FIG. 25 is a flowchart of a subroutine for detecting a rest portion at the head of a song on a measure counter to achieve consistency with the measure number on a musical score, executed in step 18_5 of the main routine shown in FIG. 18.

[0075]

In step 25_1, time from the head of a song to the first sound to be produced (rest time) is detected, and the value is set in a work register RestTime. In step 25_2, it is determined whether or not RestTime is less than 3. If it is less than 3, the process proceeds to step 25_3, and if it is not, the process proceeds to step 25_5. In step 25_3, Measure is set to '0', and the routine exits.

[0076]

In step 25_4, it is determined whether or not RestTime is less than 5. If it is less than 5, the process proceeds to step 25_5, and if it is not, the process proceeds to step 25_6. In step 25_5, Measure is set to '-1' as a measure number for anacrusis, and the routine exits. In step 25_6, Measure is set to '0', and the routine exits.

[0077]

FIG. 26 is a flowchart of a subroutine for displaying the number of a measure, executed in step 18_11 of the main routine shown in FIG. 18. In step 26_1, it is determined whether or not Measure is equal to or more than 0. If it is equal to or more than 0, the process proceeds to step 26_2, and if it is not, the process proceeds to step 26_3. In step 26_2, the display displays the value of Measure + 1.

[0078]

In step 26_3, the display displays the value of Measure. In the embodiment described above, sectors storing play data extending over two tracks are loaded twice, or once for each track. However, play data having already been loaded for another track and existing in a cache may not have to be loaded again, but may be considered as having been loaded into that cache, thereby reducing the time for loading. Also, in the embodiment described above, play data for a track with the smallest number of clock is simply loaded. However, the process of determining play data for which track is loaded into an empty cache (the processes in steps 18_3 - 18_8 of FIG. 18) may be modified. For example, play data for which track is loaded may be determined based not only on the total number of clocks of loaded data, but also on the traveling time of the head for a disk.

[0079]

[Effect of the Invention]

As has been described above, according to the play data reading device of the present invention, an internal memory being (or having just been) empty is not loaded fixedly with play data for a corresponding track, but with play data for a track with the smallest number of total clocks of play data being loaded, or with such play data selected additionally based on the position of the head of the disk device for access. This increases the use efficiency of the internal memories, allowing accurate performance with respect to the clock with a small number of internal memories.

[Brief Description of the Drawings]

FIG. 1 is a block diagram showing the configuration of an embodiment of a play data reading device of the present invention.

FIG. 2 is a flowchart showing the basic concept of a data transfer process in this embodiment.

FIG. 3 is a diagram illustrating the data flow and showing the basic concept of the data transfer process in this embodiment.

FIG. 4 shows the data structure of play data for a song about to be played, stored in a floppy disk.

FIG. 5 schematically shows changes in values of various flags and registers, used in a data transfer routine.

FIG. 6 schematically shows changes in values of various flags and registers, used in the data transfer routine.

FIG. 7 schematically shows changes in values of various flags and registers, used in the data transfer routine.

FIG. 8 schematically shows changes in values of various flags and registers, used in the data transfer routine.

FIG. 9 schematically shows changes in values of various flags and registers, used in the data transfer routine.

FIG. 10 schematically shows changes in values of various flags and registers, used in the data transfer routine.

FIG. 11 schematically shows changes in values of various flags and registers, used in the data transfer routine.

FIG. 12 schematically shows changes in values of various flags and registers, used in the data transfer routine.

FIG. 13 schematically shows changes in values of various flags and registers, used in the data transfer routine.

FIG. 14 schematically shows changes in values of various flags and registers, used in the data transfer routine.

FIG. 15 shows the format of play data stored in a floppy disk.

FIG. 16 shows the format of play data.

FIG. 17 shows the detail of a data information portion included in play data.

FIG. 18 is a flowchart of a main routine.

FIG. 19 is a flowchart of a subroutine for preparation for play of a song indicated by Song.

FIG. 20 is a flowchart of a subroutine for setting next play data.

FIG. 21 is a flowchart of a data loading subroutine for loading play data for a track.

FIG. 22 is a flowchart of a clock interrupt routine.

FIG. 23 is a flowchart of a subroutine for outputting play data to a MIDI OUT terminal, updating the tempo and beat, and detecting the end of a song, depending on the type of that play data.

FIG. 24 is a flowchart of a subroutine for stopping performance.

FIG. 25 is a flowchart of a subroutine for detecting a rest portion at the head of a song to achieve consistency with the measure number on a musical score.

FIG. 26 is a flowchart of a subroutine for displaying the number of a measure.

FIG. 27 is a flowchart showing a method of loading play data according to a conventional approach.

FIG. 28 is a diagram illustrating the conventional approach.

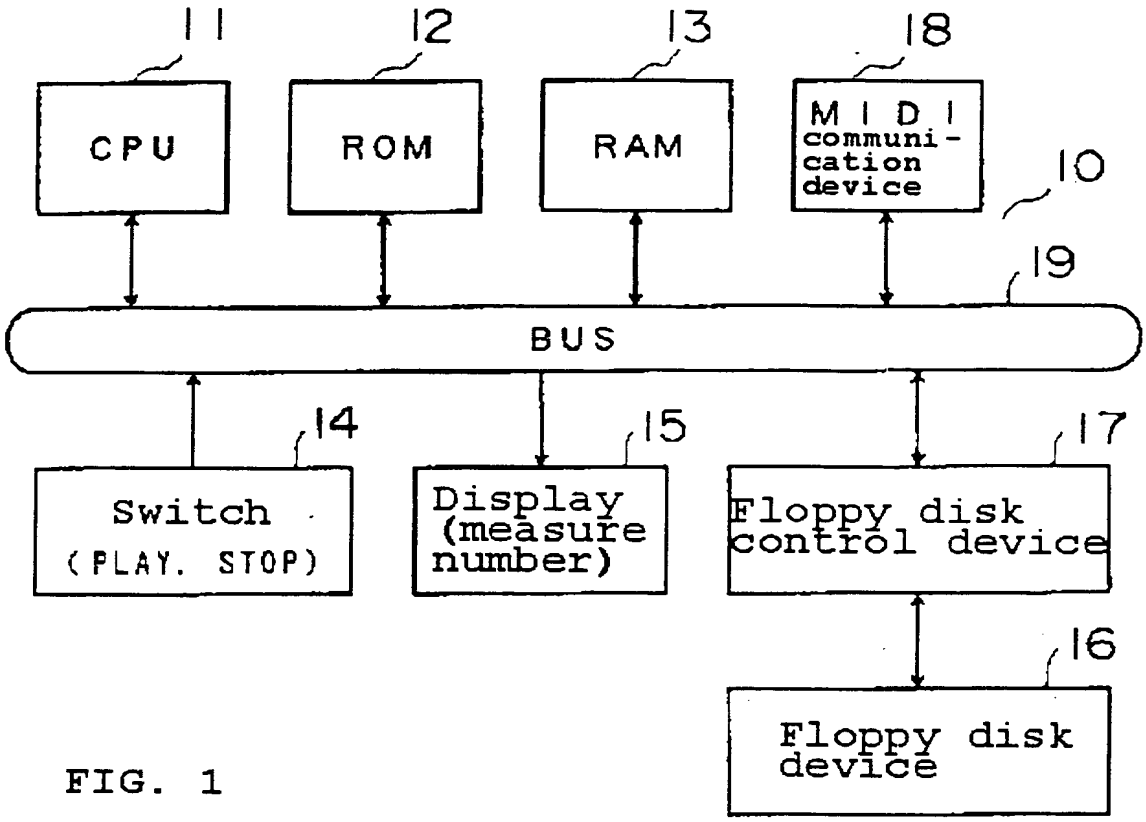
11
12
13
18
CPU
ROM
RAM
MIDI communi-cation device
10
19
BUS
14
15
17
Switch (PLAY. STOP)
Display (measure number)
Floppy disk control device
16
Floppy disk device

FIG. 1

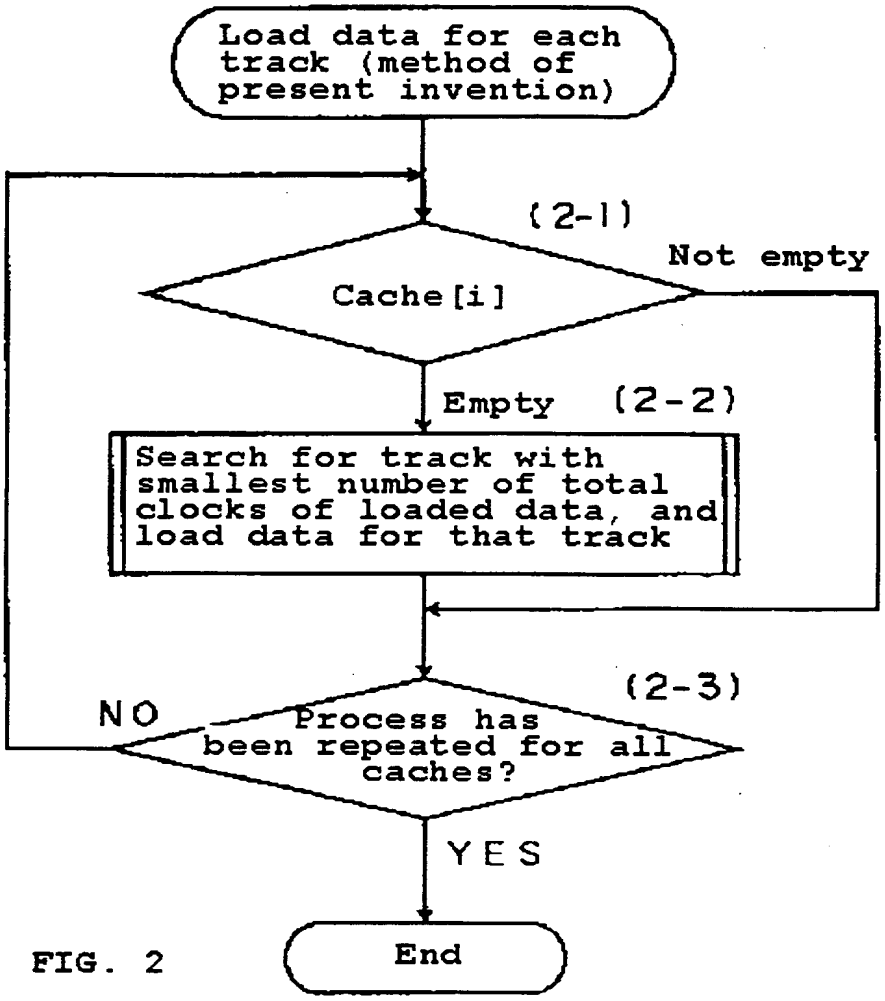
Load data for each track (method of present invention)
(2-1)
Cache[i]
Not empty
Empty
(2-2)
Search for track with smallest number of total clocks of loaded data, and load data for that track
(2-3)
NO
Process has been repeated for all caches?
YES
End

FIG. 2

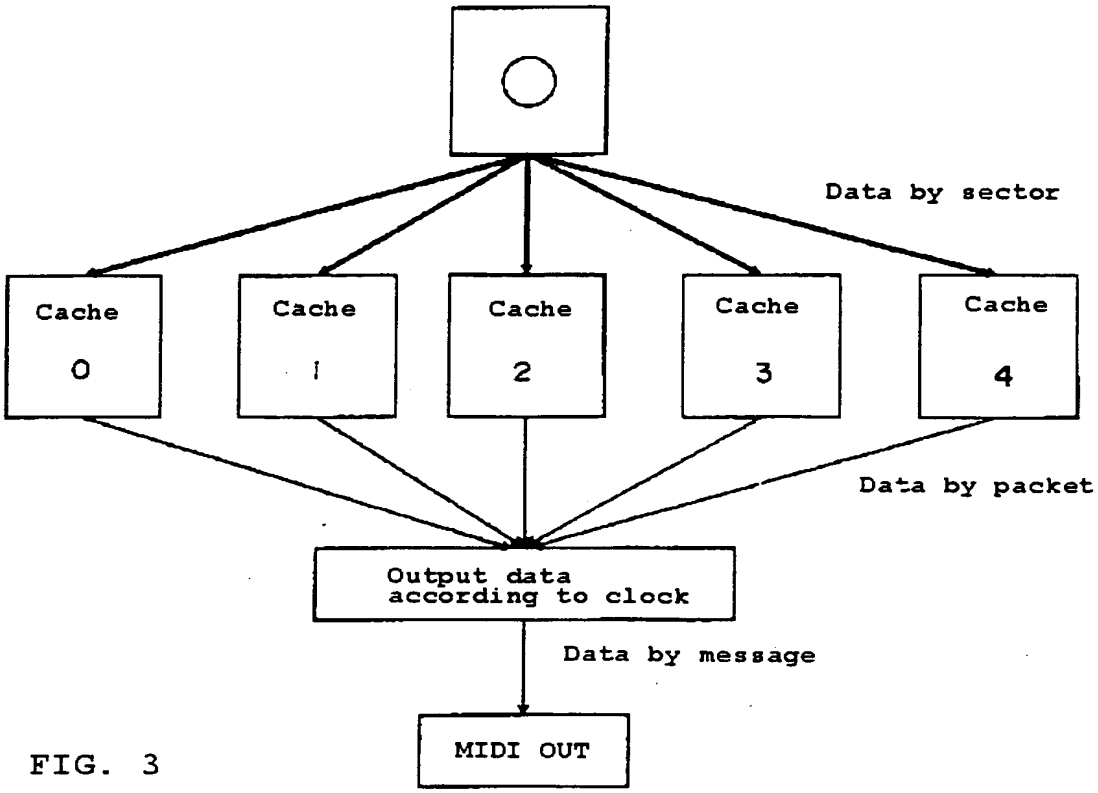

FIG. 3

Track number	track0		track1				track2		
Number of clocks in sector	150	10	10	50	50	50	20	100	40
Sector number	sector0	sector1		sector2	sector3	sector4		sector5	sector6

FIG. 4

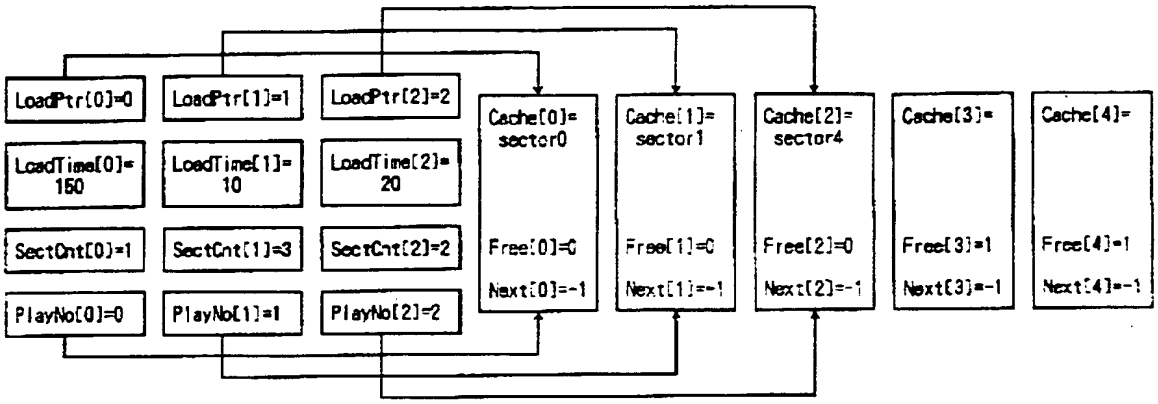

FIG. 5

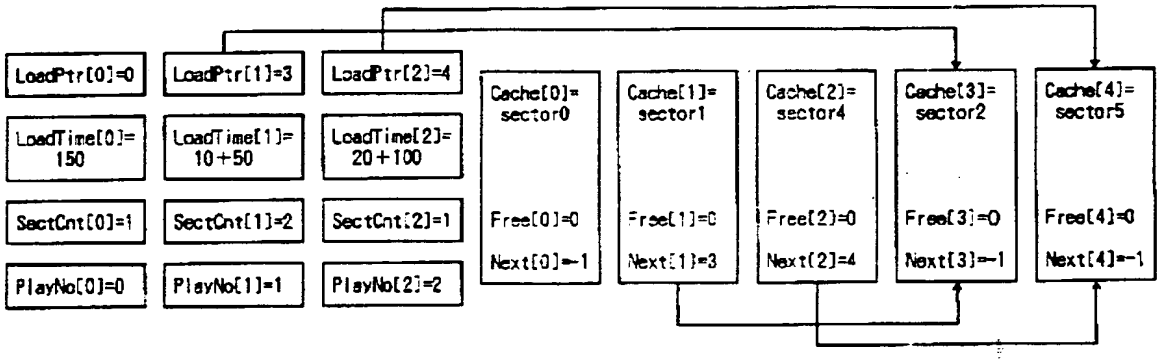

FIG. 6

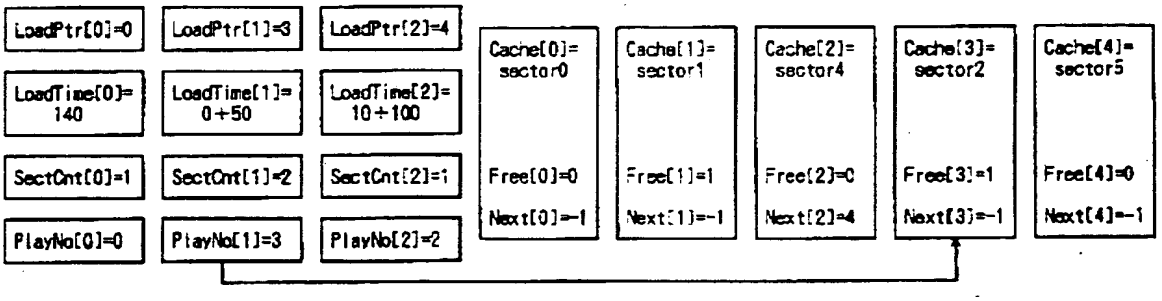

FIG. 7

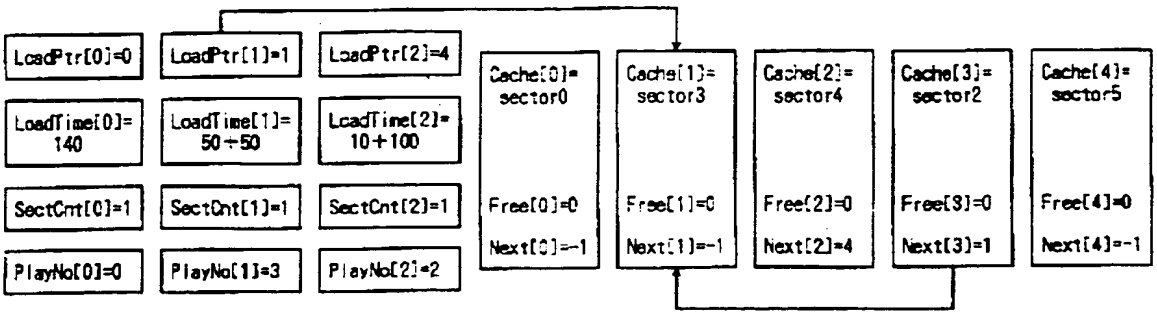

FIG. 8

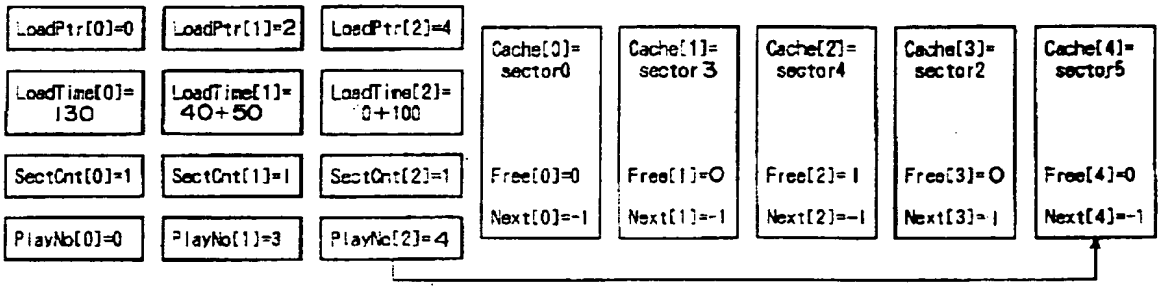

FIG. 9

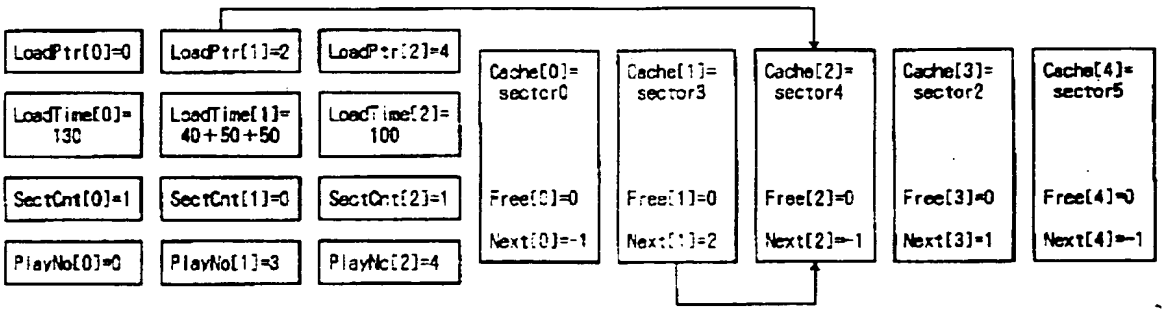

FIG. 10

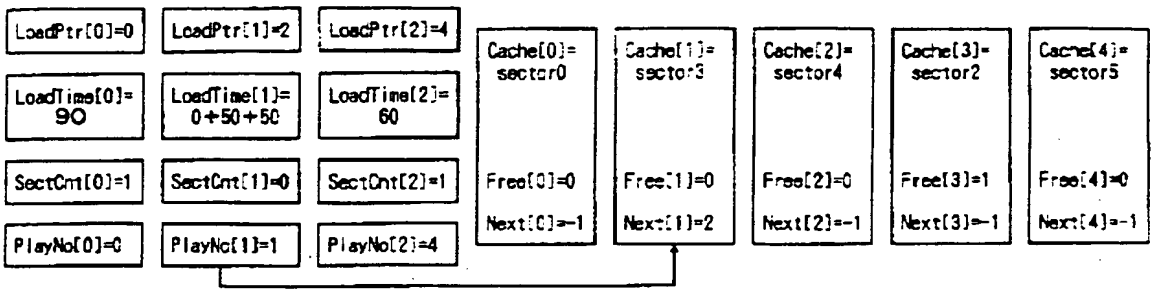

FIG. 11

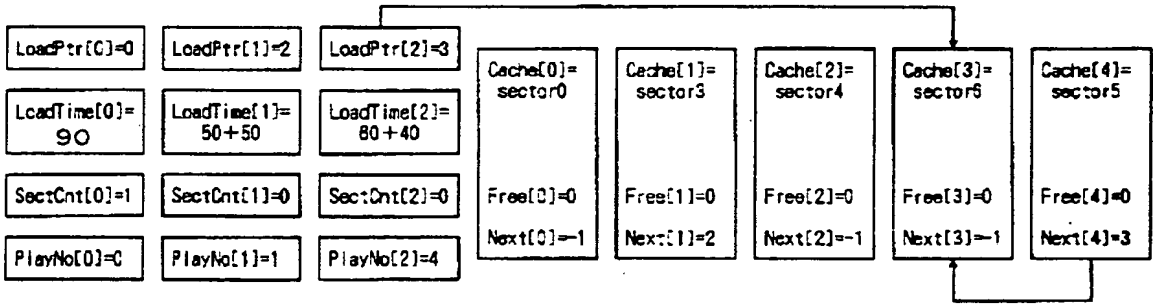

FIG. 12

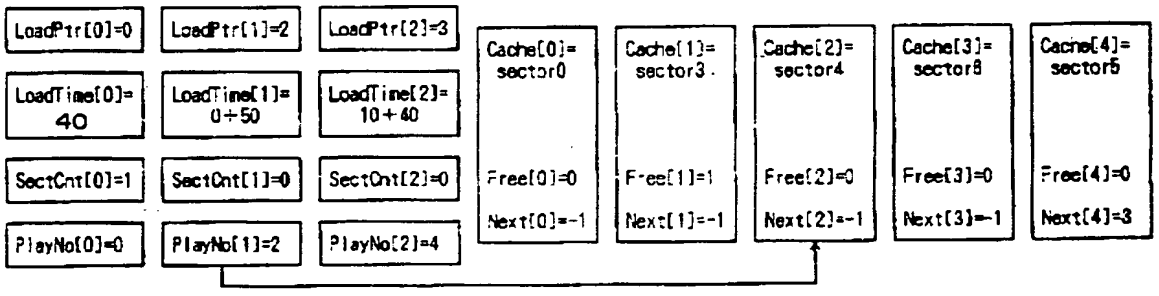

FIG. 13

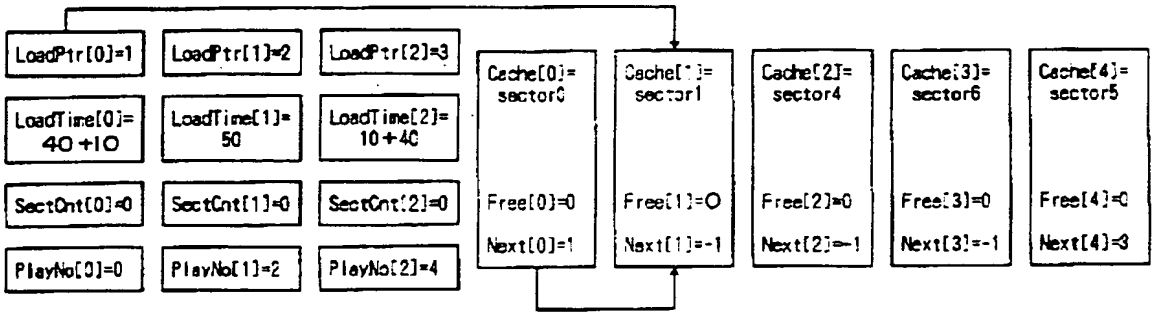

FIG. 14

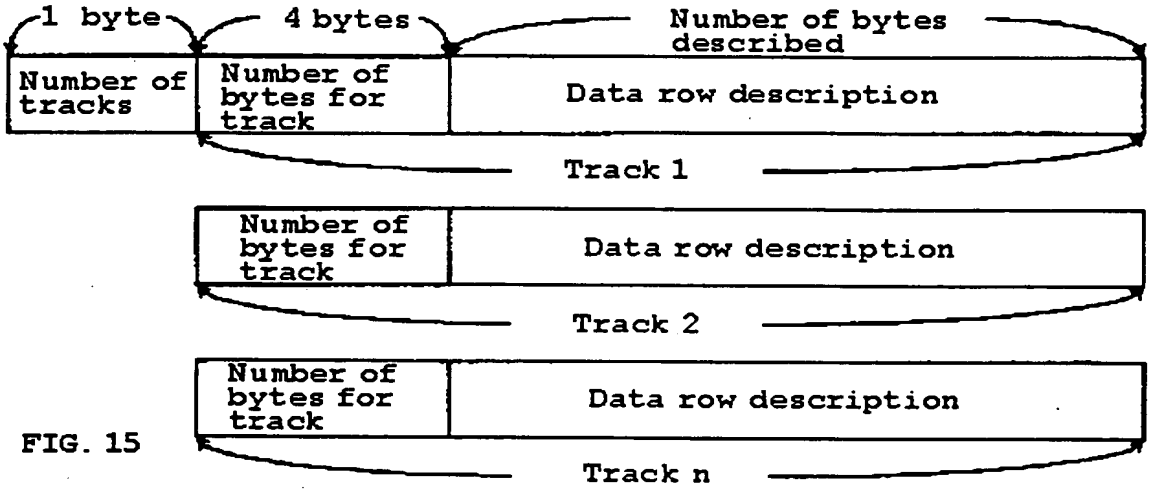

FIG. 15

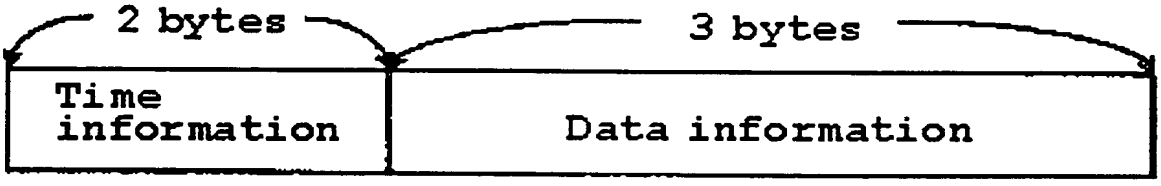

FIG. 16

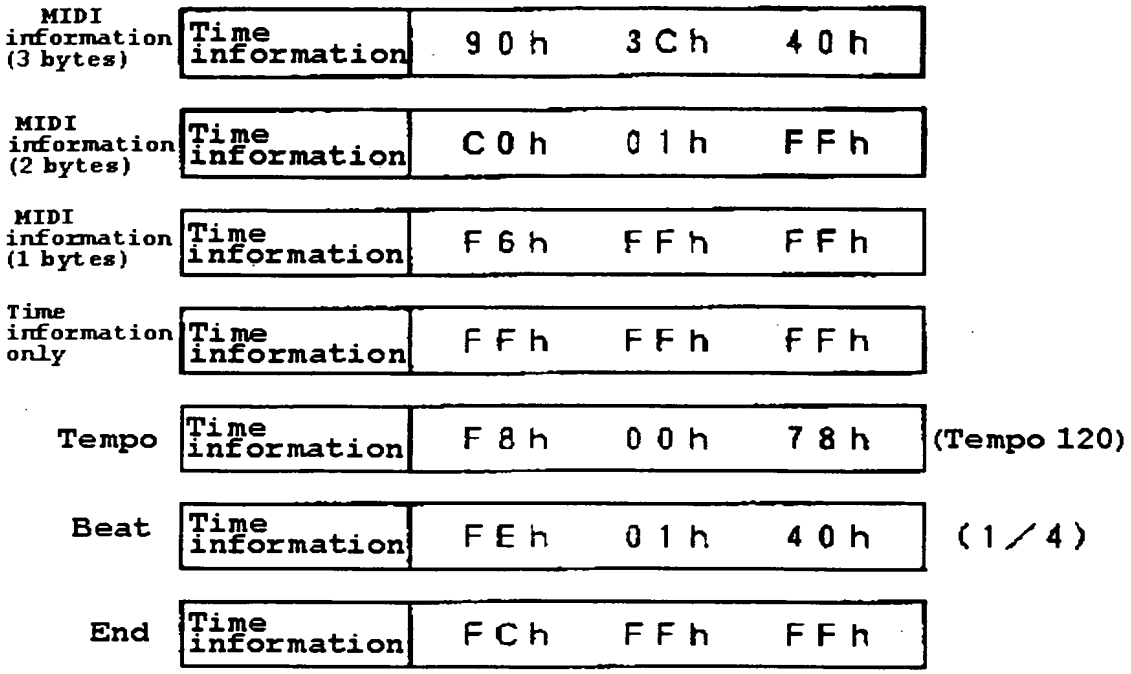
MIDI information (3 bytes)
Time information
90h 3Ch 40h
MIDI information (2 bytes)
Time information
C0h 01h FFh
MIDI information (1 bytes)
Time information
F6h FFh FFh
Time information only
Time information
FFh FFh FFh
Tempo
Time information
F8h 00h 78h
(Tempo 120)
Beat
Time information
FEh 01h 40h
(1/4)
End
Time information
FCh FFh FFh

FIG. 17

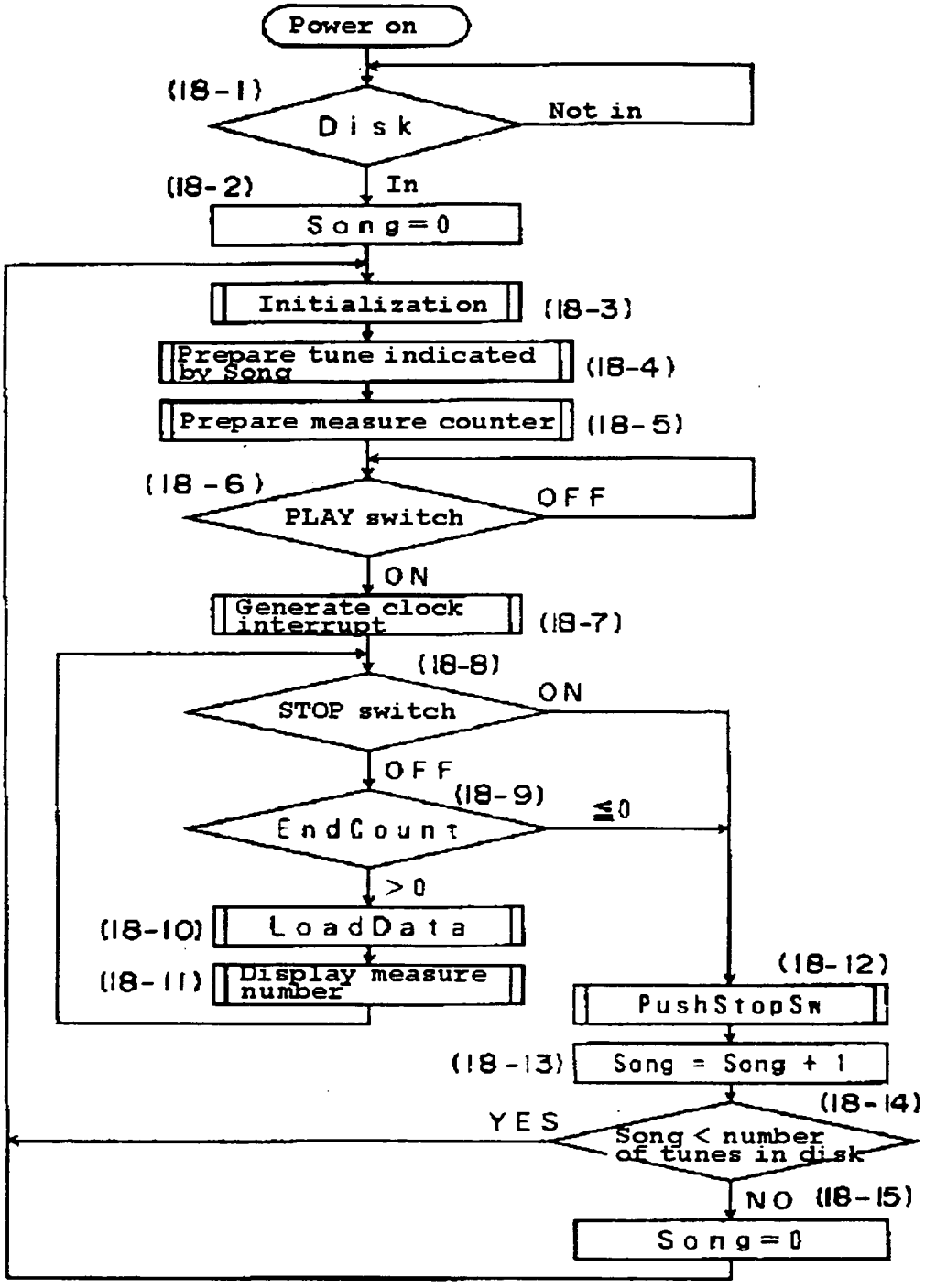
Power on
(18-1)
Disk
Not in
(18-2)
In
Song=0
Initialization
(18-3)
Prepare tune indicated by Song
(18-4)
Prepare measure counter
(18-5)
(18-6)
PLAY switch
OFF
ON
Generate clock interrupt
(18-7)
(18-8)
STOP switch
ON
OFF
(18-9)
EndCount
≦0
>0
(18-10)
LoadData
(18-11)
Display measure number
(18-12)
PushStopSw
(18-13)
Song = Song + 1
(18-14)
YES
Song < number of tunes in disk
NO
(18-15)
Song=0

FIG. 18

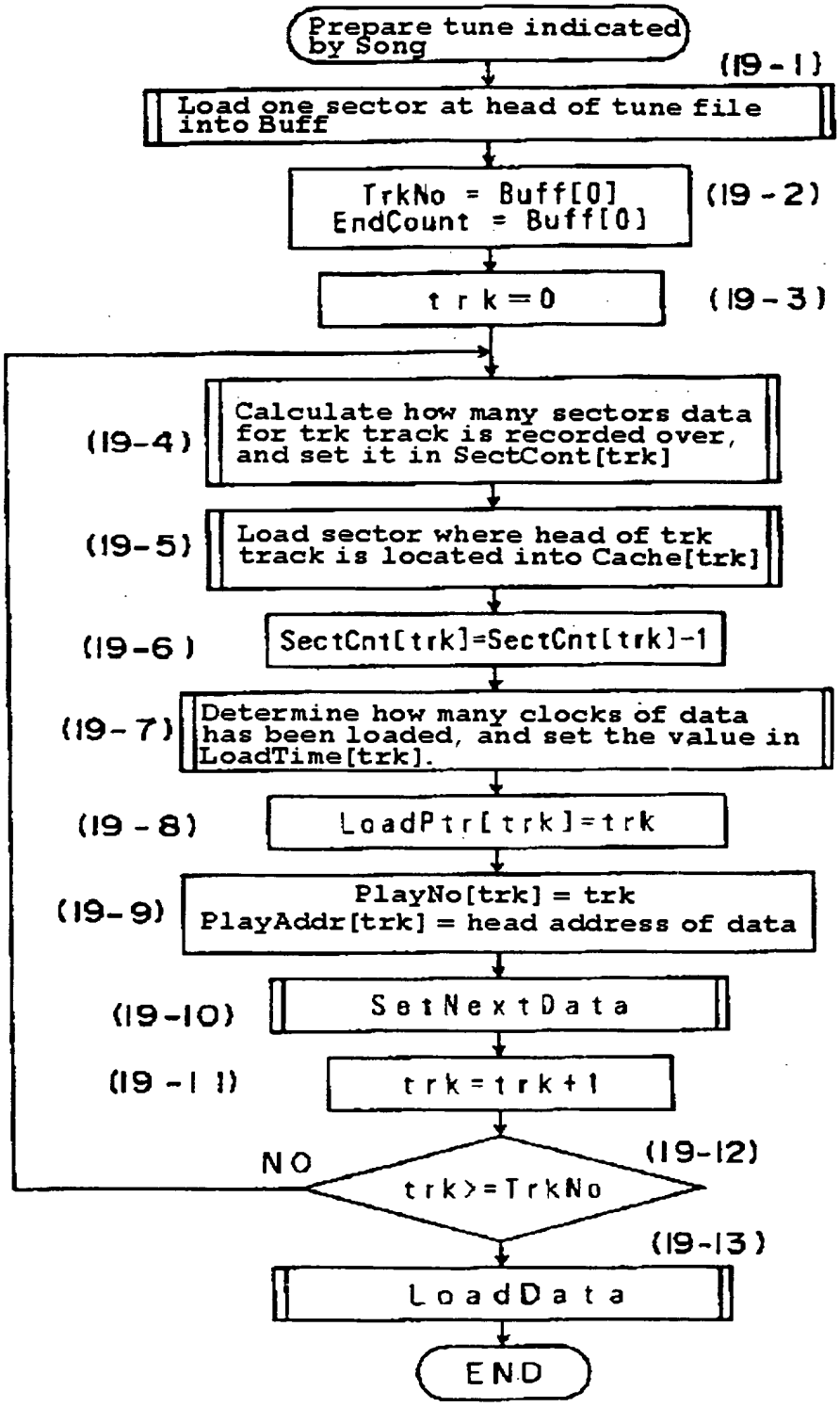
Prepare tune indicated by Song
(19-1)
Load one sector at head of tune file into Buff
TrkNo = Buff[0]
EndCount = Buff[0]
(19-2)
trk=0
(19-3)
(19-4)
Calculate how many sectors data for trk track is recorded over, and set it in SectCont[trk]
(19-5)
Load sector where head of trk track is located into Cache[trk]
(19-6)
SectCnt[trk]=SectCnt[trk]-1
(19-7)
Determine how many clocks of data has been loaded, and set the value in LoadTime[trk].
(19-8)
LoadPtr[trk]=trk
(19-9)
PlayNo[trk] = trk
PlayAddr[trk] = head address of data
(19-10)
SetNextData
(19-11)
trk=trk+1
NO
(19-12)
trk>=TrkNo
(19-13)
LoadData
END

FIG. 19

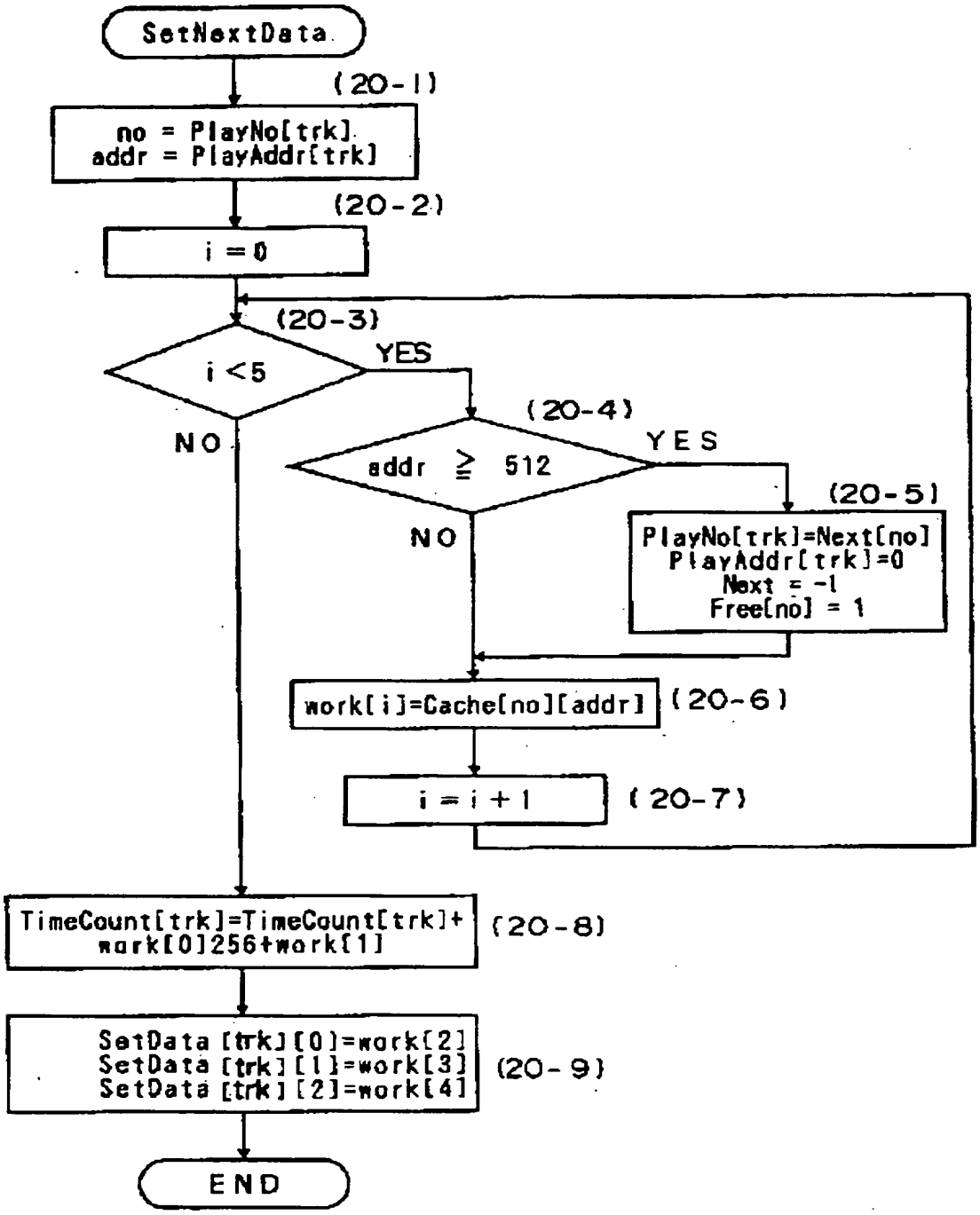
SetNextData
(20-1)
no = PlayNo[trk]
addr = PlayAddr[trk]
(20-2)
i = 0
(20-3)
i < 5
YES
NO
(20-4)
addr ≧ 512
YES
NO
(20-5)
PlayNo[trk]=Next[no]
PlayAddr[trk]=0
Next = -1
Free[no] = 1
work[i]=Cache[no][addr]
(20-6)
i = i + 1
(20-7)
TimeCount[trk]=TimeCount[trk]+
work[0]256+work[1]
(20-8)
SetData [trk][0]=work[2]
SetData [trk][1]=work[3]
SetData [trk][2]=work[4]
(20-9)
END

FIG. 20

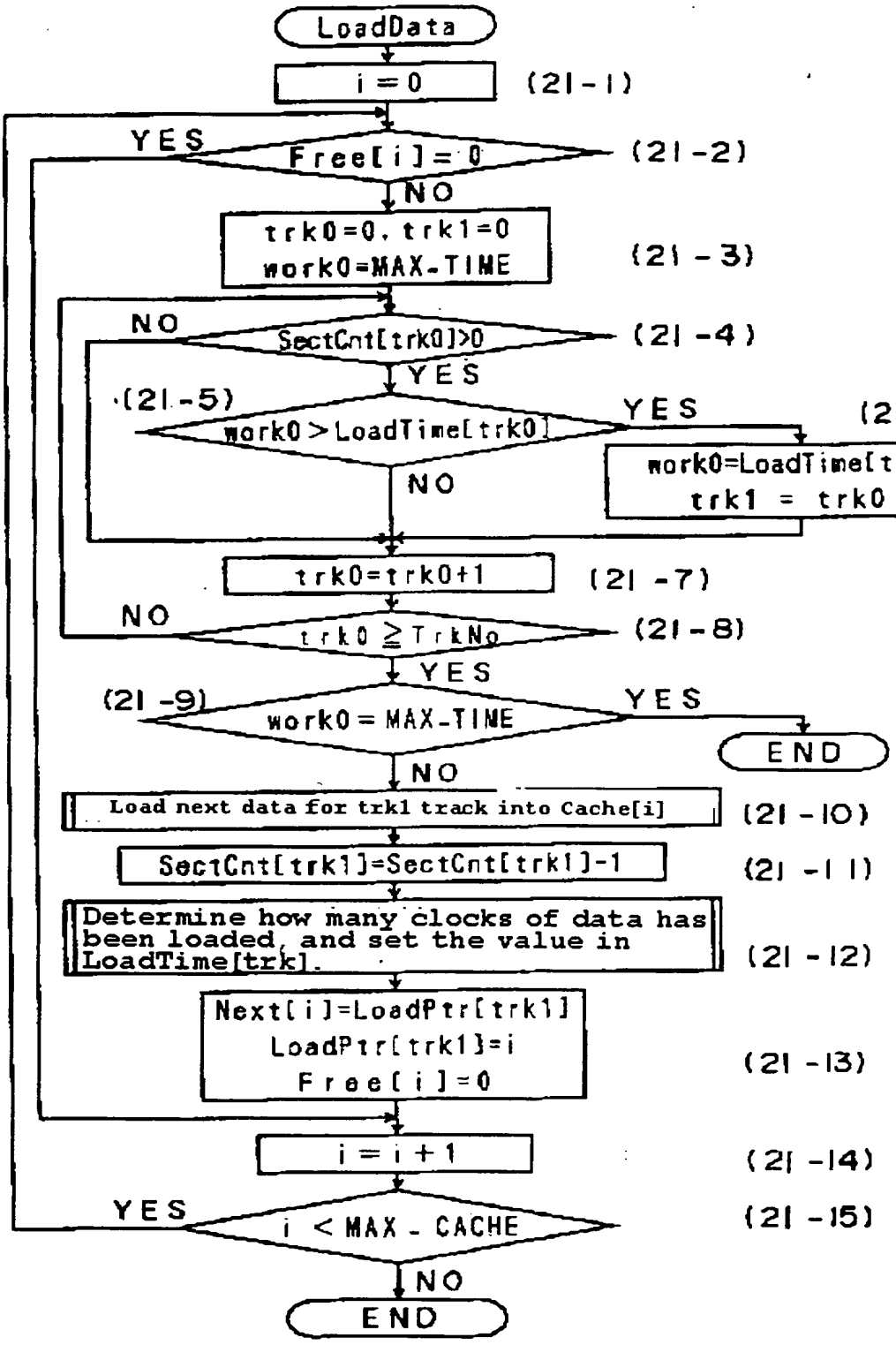
LoadData
i = 0
(21-1)
YES
Free[i] = 0
(21-2)
NO
trk0=0, trk1=0
work0=MAX-TIME
(21-3)
NO
SectCnt[trk0]>0
(21-4)
YES
(21-5)
work0 > LoadTime[trk0]
YES
(21-6)
work0=LoadTime[trk0]
trk1 = trk0
NO
trk0=trk0+1
(21-7)
NO
trk0 ≧ TrkNo
(21-8)
YES
(21-9)
work0 = MAX-TIME
YES
END
NO
Load next data for trk1 track into Cache[i]
(21-10)
SectCnt[trk1]=SectCnt[trk1]-1
(21-11)
Determine how many clocks of data has been loaded, and set the value in LoadTime[trk].
(21-12)
Next[i]=LoadPtr[trk1]
LoadPtr[trk1]=i
Free[i]=0
(21-13)
i = i + 1
(21-14)
YES
i < MAX - CACHE
(21-15)
NO
END

FIG. 21

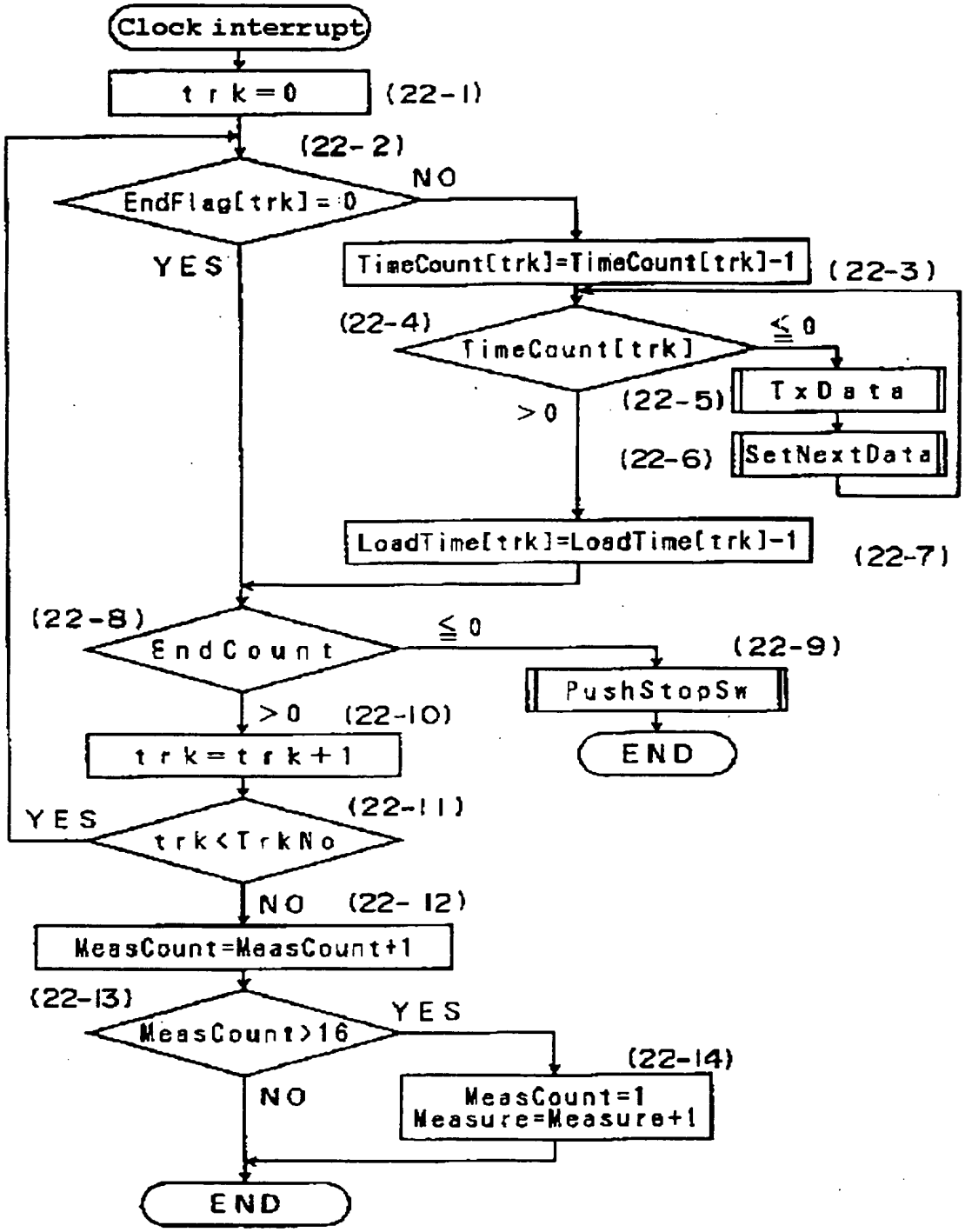
Clock interrupt
trk=0
(22-1)
(22-2)
EndFlag[trk]=0
NO
YES
TimeCount[trk]=TimeCount[trk]-1
(22-3)
(22-4)
TimeCount[trk]
≦0
>0
(22-5)
TxData
(22-6)
SetNextData
LoadTime[trk]=LoadTime[trk]-1
(22-7)
(22-8)
EndCount
≦0
(22-9)
PushStopSw
END
>0
(22-10)
trk=trk+1
YES
(22-11)
trk<TrkNo
NO
(22-12)
MeasCount=MeasCount+1
(22-13)
MeasCount>16
YES
(22-14)
NO
MeasCount=1
Measure=Measure+1
END

FIG. 22

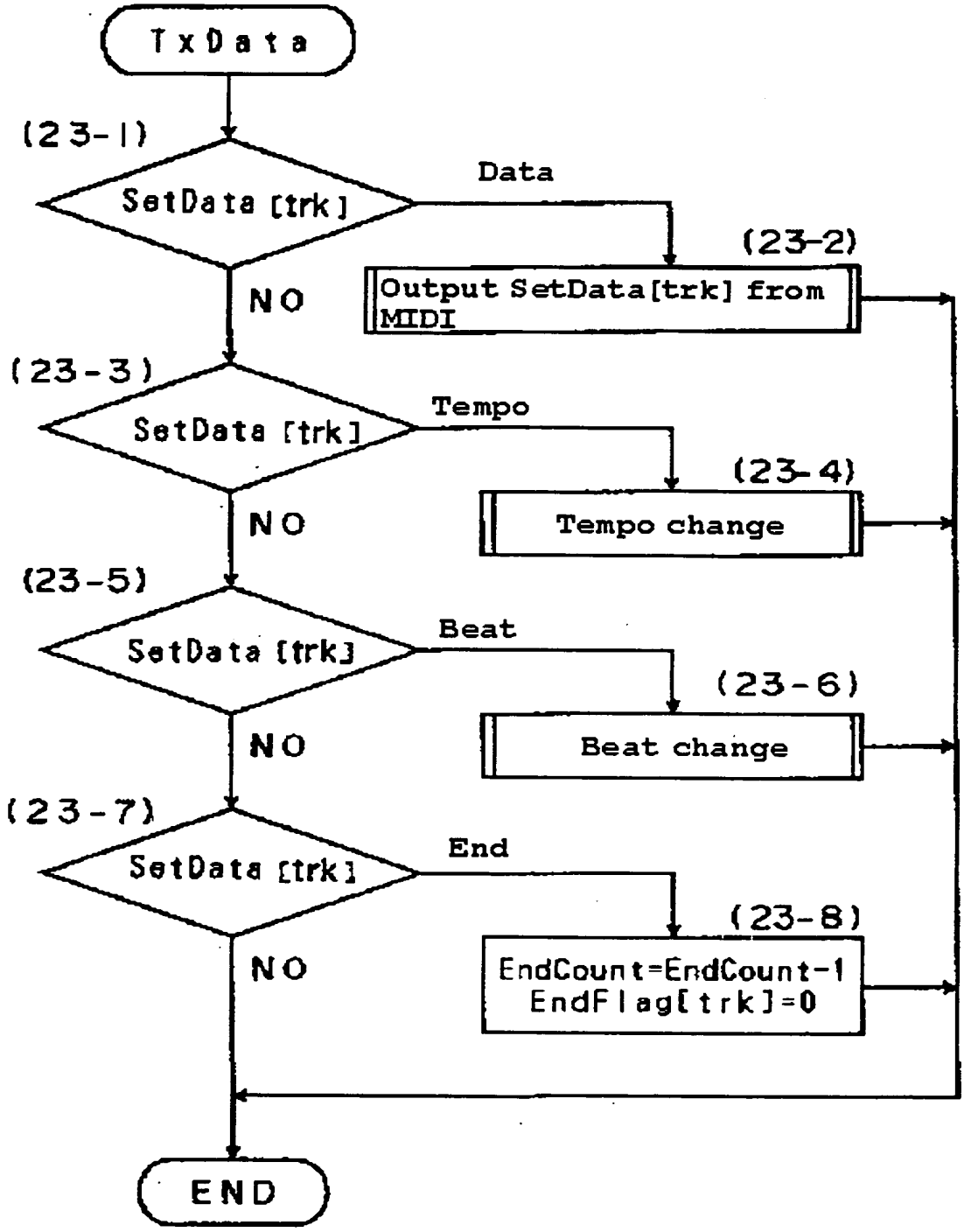
TxData
(23-1)
SetData [trk]
Data
(23-2)
Output SetData[trk] from MIDI
NO
(23-3)
SetData [trk]
Tempo
(23-4)
Tempo change
NO
(23-5)
SetData [trk]
Beat
(23-6)
Beat change
NO
(23-7)
SetData [trk]
End
(23-8)
EndCount=EndCount-1
EndFlag[trk]=0
NO
END

FIG. 23

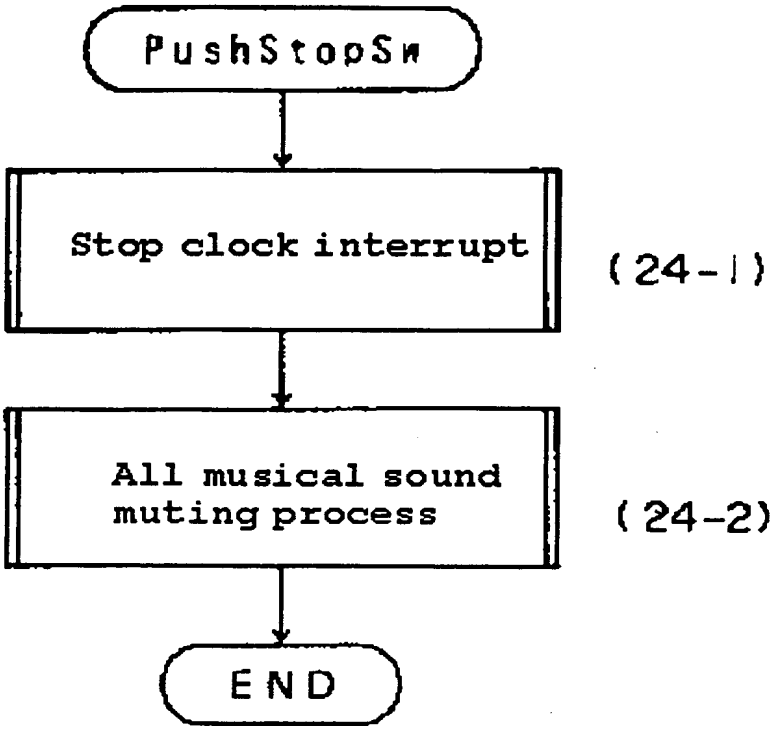
PushStopSw
Stop clock interrupt
(24-1)
All musical sound muting process
(24-2)
END

FIG. 24

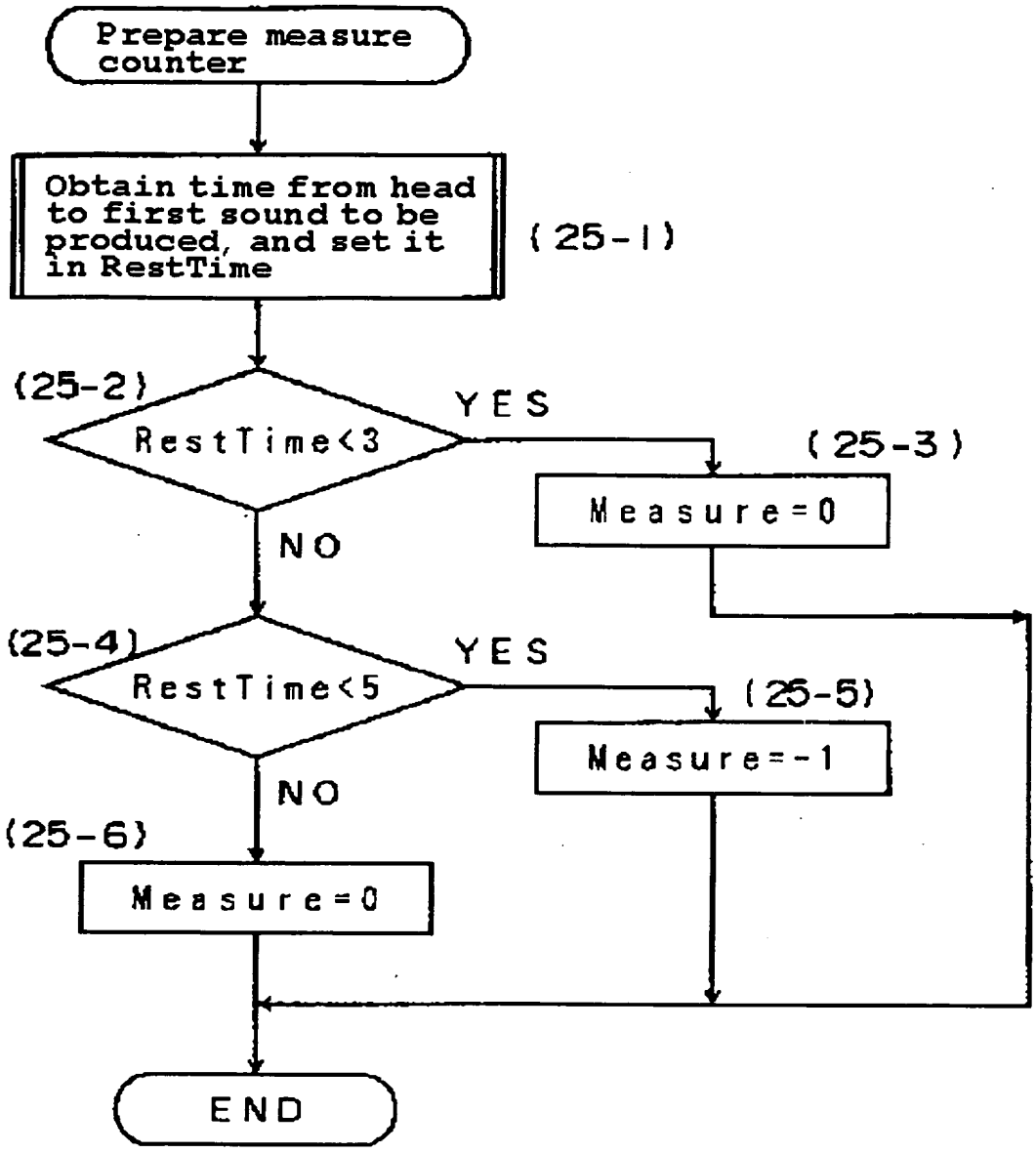
Prepare measure counter
Obtain time from head to first sound to be produced, and set it in RestTime
(25-1)
(25-2)
RestTime<3
YES
(25-3)
Measure=0
NO
(25-4)
RestTime<5
YES
(25-5)
Measure=-1
NO
(25-6)
Measure=0
END

FIG. 25

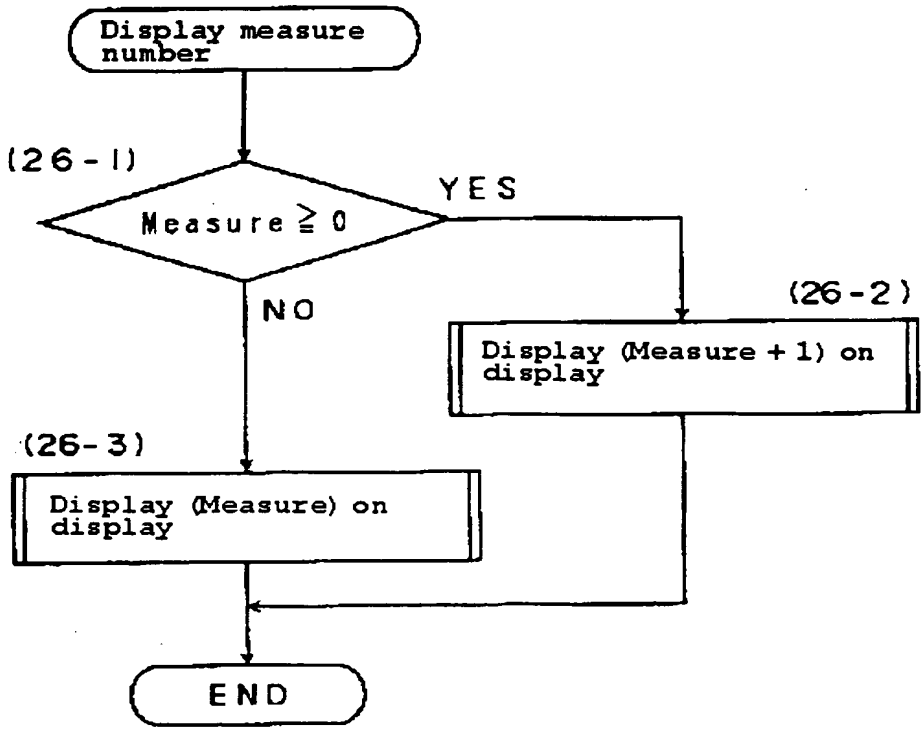
Display measure number
(26-1)
Measure ≧ 0
YES
NO
(26-2)
Display (Measure + 1) on display
(26-3)
Display (Measure) on display
END

FIG. 26

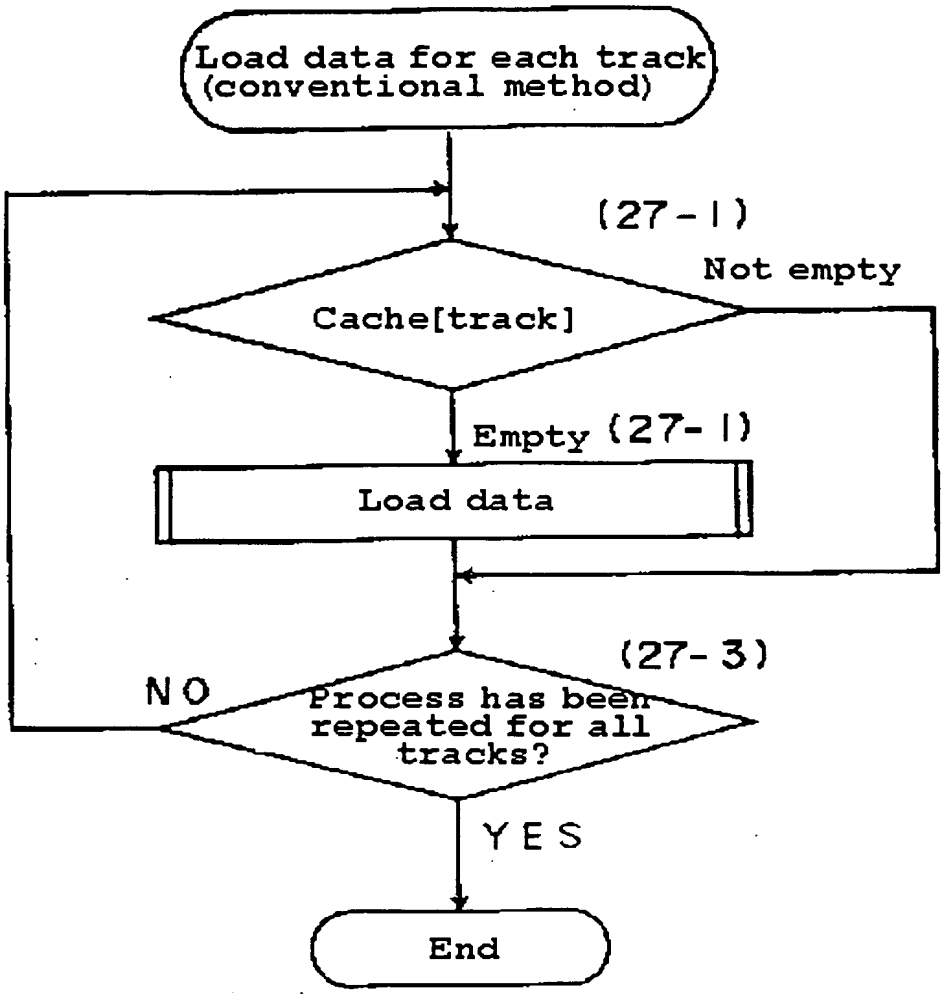
Load data for each track (conventional method)
(27-1)
Not empty
Cache[track]
Empty (27-1)
Load data
(27-3)
NO
Process has been repeated for all tracks?
YES
End

FIG. 27

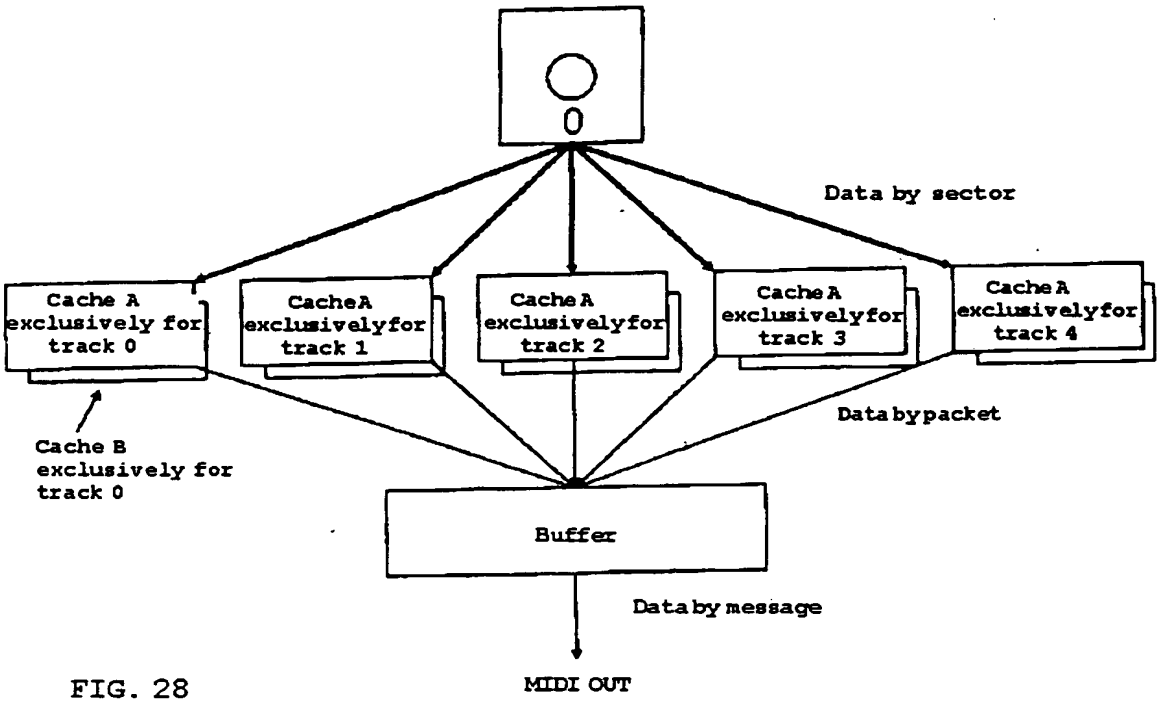
0
Data by sector
Cache A exclusively for track 0
Cache A exclusively for track 1
Cache A exclusively for track 2
Cache A exclusively for track 3
Cache A exclusively for track 4
Cache B exclusively for track 0
Data by packet
Buffer
Data by message
MIDI OUT

FIG. 28